Les arbres-B

Géraldine Del Mondo, Nicolas Delestre

Fondé sur le cours de M. Michel Mainguenaud - Dpt ASI

INSA ROUEN NORMANDIE

Plan...

INSA ROUEN NORMANDIE

Contexte

L'arbre-B (où *B-Tree* en anglais) est une SDD utilisée dans les domaines des :

- systèmes de gestion de fichiers : ReiserFS (version modifiée des arbres-B) ou Btrfs (*B-Tree file system*)
- bases de données : gestion des index

L'arbre-B reprend le concept d'ABR équilibré mais en stockant dans un nœud k valeurs (nommées clés dans le contexte des arbres-B) et en référençant $k + 1$ sous arbres :

- « minimise la taille de l'arbre et réduit le nombre d'opérations d'équilibrage » (Wikipédia)
- utile pour un stockage sur une unité de masse

INSA ROUEN NORMANDIE

Arbre-B

Définition ARBRE-B

Un arbre-B d'ordre m est un arbre tel que :

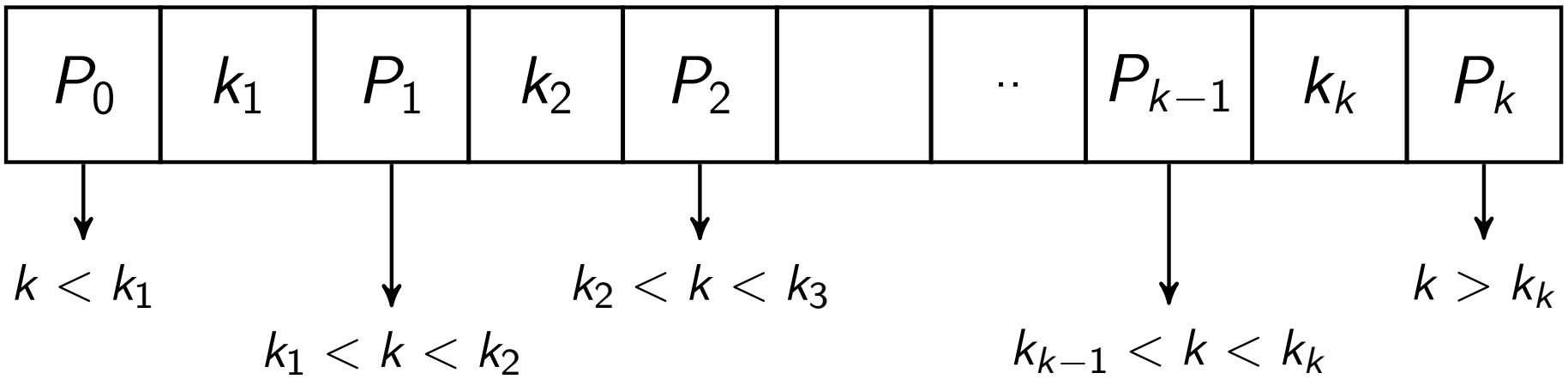
1. Chaque nœud contient k clés triées avec : $m \leq k \leq 2m$ (nœud non racine) et $1 \leq k \leq 2m$ (nœud racine).
2. Chaque chemin de la racine à une feuille est de même longueur à 1 près
3. Un nœud est :
   - Soit terminal (feuille)
   - Soit possède $(k+1)$ fils tels que les clés du ième fils ont des valeurs comprises entre les valeurs du $(i-1)$ème et $i$ème clés du père

INSA ROUEN NORMANDIE

Structure d'un nœud

Définition Nœud

- k clés triées
- $k+1$ pointeurs tels que :
  - Tous sont différents de NIL si le nœud n'est pas une feuille
  - Tous à NIL si le nœud est une feuille



Capacité

- **Nombre de clés**

  Arbre-B d'ordre m et de hauteur h :

→ Nombre de clé(s) minimal $= 2 * (m+1)^h - 1$

→ Nombre de clés maximal $= (2 * m + 1)^{h+1} - 1$

  EXEMPLE : $m = 100$, $h = 2$: Nombre maximal de clés $\approx 8\ 000\ 000$

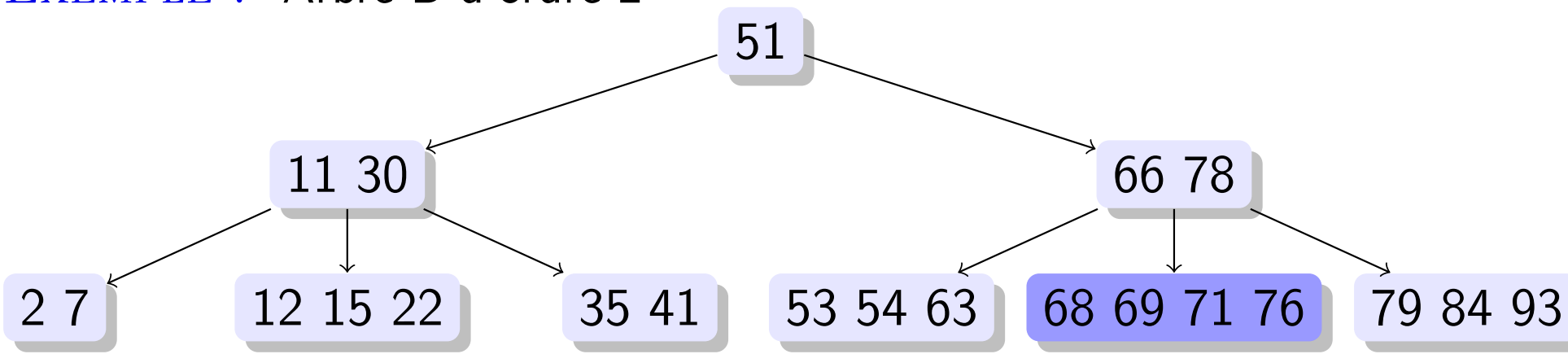
- **Stockage sur disque**

→ Un noeud = Un bloc (ensemble de secteurs)

INSA ROUEN NORMANDIE

Exemple

EXEMPLE : Arbre-B d'ordre 2



- Chaque nœud, sauf la racine contient k clés avec $2 \leq k \leq 4$
- La racine contient k clé(s) avec $1 \leq k \leq 4$

Conception

Conception détaillée

Type ArbreB = ^Noeud
Type Noeud = **Structure**
    nbCles : **NaturelNonNul**
    cles : **Tableau**[1..MAX] **de** Valeur
    sousArbres : **Tableau**[0..MAX] **de** ArbreB
finstructure

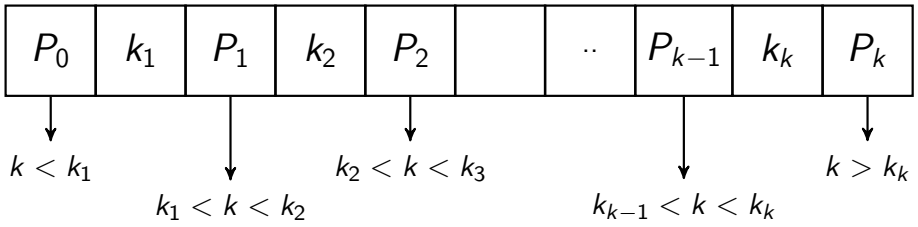
. . . tel que le type `Valeur` possède un ordre total

Information

Contrairement au premier cours sur les SDD, nous utiliserons ici aucune fonction/procédure d'encapsulation

INSA ROUEN NORMANDIE

Recherche d'un élément de clé c 1 / 3



Principe

À partir de la racine, pour chaque nœud examiné :

- La clé C est présente (recherche qui peut être dichotomique) $\rightarrow$ succès
- $C < k_1 \rightarrow$ recherche dans le sous-arbre le plus à gauche (via le pointeur P_0)
- $C > k_k \rightarrow$ recherche dans le sous-arbre le plus à droite (via le pointeur P_k)
- $k_i < C < k_{i+1}$ (recherche qui peut être dichotomique) $\rightarrow$ recherche dans le sous-arbre (via le pointeur P_i)
- Si l'arbre est vide (pointeur vaut NIL) $\rightarrow$ échec

Recherche d'un élément de clé c 2 / 3

```
fonction estPresent (a : ArbreB, c : Valeur) : Booleen
debut
    si a=NIL alors
        retourner FAUX
    sinon
        si c<a^.cles[1] alors
            retourner estPresent(a^.sousArbres[0],c)
        sinon
            si c>a^.cles[a^.nbCles] alors
                retourner estPresent(a^.sousArbres[a^.nbCles],c)
            sinon
                rechercherDansNoeud(a^,c,res,ssArbre)
                si res alors
                    retourner VRAI
                sinon
                    retourner estPresent(ssArbre,c)
                finsi
            finsi
        finsi
    finsi
fin
```

Recherche d'un élément de clé c 3 / 3

```
procédure rechercherDansNoeud (E n : Noeud, c : Valeur, S estPresent : Booleen, sousArbre : ArbreB)
    Déclaration   g,d,m : NaturelNonNul
debut
    g ← 1
    d ← n.nbCles
    tant que g≠d faire
        m ← (g+d) div 2
        si n.cles[m]≥c alors
            d ← m
        sinon
            g ← m+1
        finsi
    fintantque
    si n.cles[g]=c alors
        estPresent ← VRAI
        sousArbre ← NIL
    sinon
        estPresent ← FAUX
        sousArbre ← n.sousArbres[g-1]
    finsi
fin
```


Remarque

g et d référencent la position de l'élément supérieur ou égal à la valeur recherchée

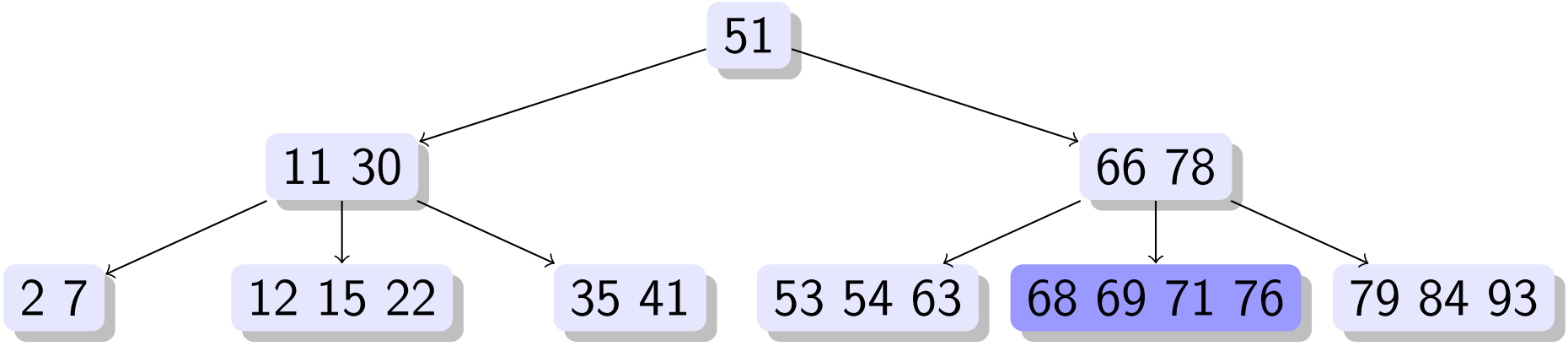
Insertion dans un arbre-B d'ordre m

Principe

1. L'insertion se fait récursivement au niveau des feuilles
2. Si un nœud a alors plus $2m + 1$ clés, il y a éclatement du nœud et remontée (grâce à la récursivité) de la clé médiane au niveau du père
3. Il y a augmentation de la hauteur de l'arbre lorsque la racine se retrouve avec $2m + 1$ clés
   $\hookrightarrow$ l'augmentation de la hauteur de l'arbre se fait donc au niveau de la racine !

Exemple : cas d'un nœud plein 1 / 2

EXEMPLE : Insertion de **75** ?



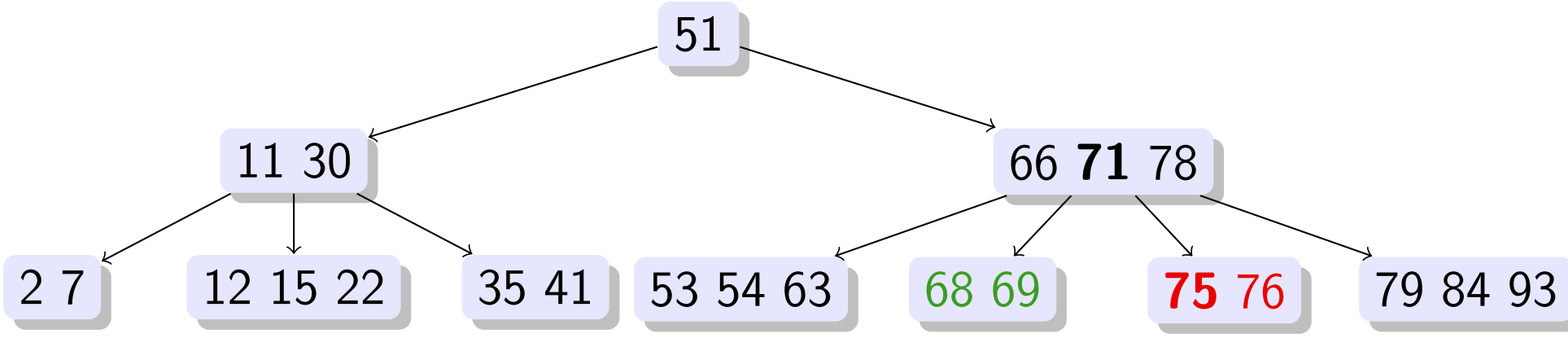
Rappel : ici nombre de clés par nœud ≤ 4

Méthode

1. Eclatement du nœud en deux :
   - Les (deux) plus petites clés restent dans le nœud
   - Les (deux) plus grandes clés sont insérées dans un nouveau nœud
2. Remontée de la clé médiane dans le nœud père (e.g. ici **71**)

Exemple : cas d'un nœud plein 2 / 2

EXEMPLE : Insertion de **75** ?



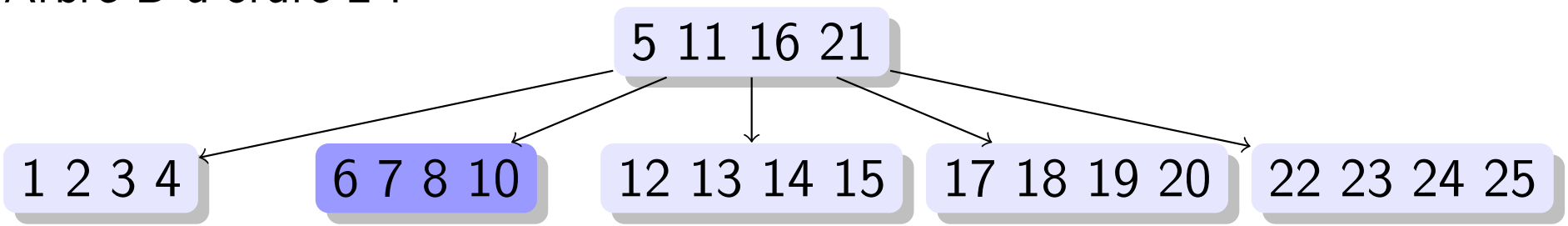
Rappel : ici nombre de clés par nœud ≤ 4

Méthode

1. Eclatement du nœud en deux :
   - Les (deux) plus petites clés restent dans le nœud
   - Les (deux) plus grandes clés sont insérées dans un nouveau nœud
2. Remontée de la clé médiane dans le nœud père (e.g. ici **71**)

Exemple : cas de l'augmentation de la hauteur 1 / 2

Arbre-B d'ordre 2 :

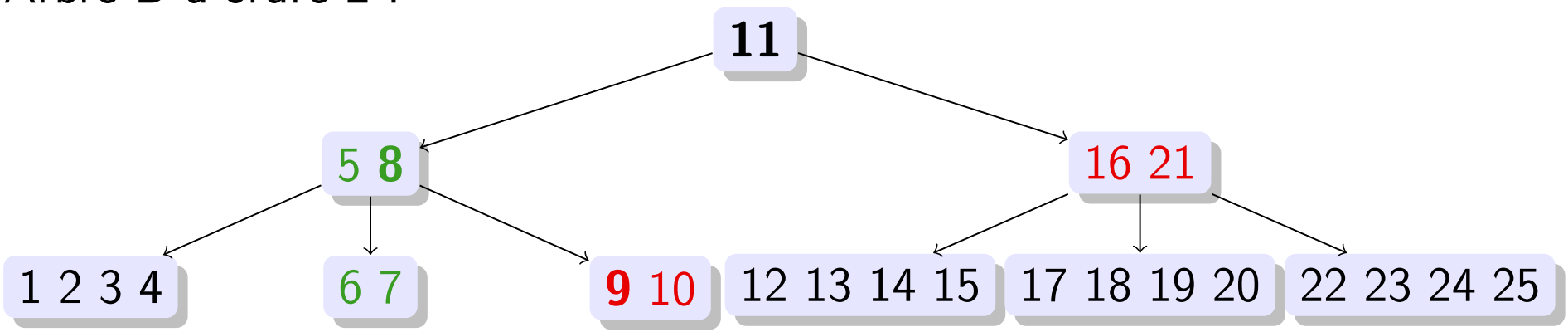


- Insertion clé **9** → Eclatement + remontée de la clé **8** au nœud père
- Remontée de la clé **8** au nœud père → Eclatement + création nouvelle racine (e.g. ici **11**)

INSA ROUEN NORMANDIE

Exemple : cas de l'augmentation de la hauteur 2 / 2

Arbre-B d'ordre 2 :



- Insertion clé **9** → Eclatement + remontée de la clé **8** au nœud père
- Remontée de la clé **8** au nœud père → Eclatement + création nouvelle racine (e.g. ici **11**)

↪ **Augmentation d'une unité de la hauteur**

INSA ROUEN NORMANDIE

Algorithme 1 / 2

Prérequis

On suppose posséder les fonctions/procédures suivantes :

- **fonction** creerFeuille **(**c :**Tableau[**1..MAX**] de** Valeur, nb : **NaturelNonNul) :** ArbreB
- **fonction** estUneFeuille **(**a : ArbreB**) : Booleen**
- **fonction** eclatement **(**a : ArbreB, ordre : **NaturelNonNul) :** ArbreB
  ⌊**précondition(s)** a^.nbCles=2*ordre+1
- **fonction** positionInsertion **(**a : ArbreB, c : Valeur**) : NaturelNonNul**
- **procédure** insererUneCleDansNoeud **(E/S** n : Noeud, **E** c : Valeur, pos : **NaturelNonNul)**
- **procédure** insererUnArbreDansNoeud **(E/S** n : Noeud, **E** a : ArbreB, pos : **NaturelNonNul)**

procédure inserer **(E/S** a : ArbreB, **E** c : Valeur, ordre : **NaturelNonNul)**
⌊**précondition(s)** 2*ordre<MAX
debut
a ← insertion(a,c,ordre)
fin

Algorithme 2 / 2

```
fonction insertion (a : ArbreB, c : Valeur, ordre : NaturelNonNul) : ArbreB
        précondition(s)     2*ordre < MAX

        Déclaration     tab : Tableau[1..MAX] de Valeur

debut
        si a = NIL alors
                tab[1] ⟵ c
                retourner creerFeuille(tab,1)
        sinon
                pos ⟵ positionInsertion(a,c)
                si estUneFeuille(a) alors
                        insererUneCleDansNoeud(a^,c,pos)
                        si a^.nbCles ≤ 2*ordre alors
                                retourner a
                        sinon
                                retourner eclatement(a, ordre)
                        finsi
                sinon
                        ssArbre ⟵ a^.sousArbres[pos-1]
                        temp ⟵ insertion(ssArbre,c,ordre)
                        si ssArbre = temp alors
                                retourner a
                        sinon
                                insererUnArbreDansNoeud(a^,temp,pos)
                                si a^.nbCles ≤ 2*ordre alors
                                        retourner a
                                sinon
                                        retourner eclatement(a, ordre)
                                finsi
                        finsi
                finsi
        finsi
fin
```

Supression dans un arbre-B d'ordre m

Principe

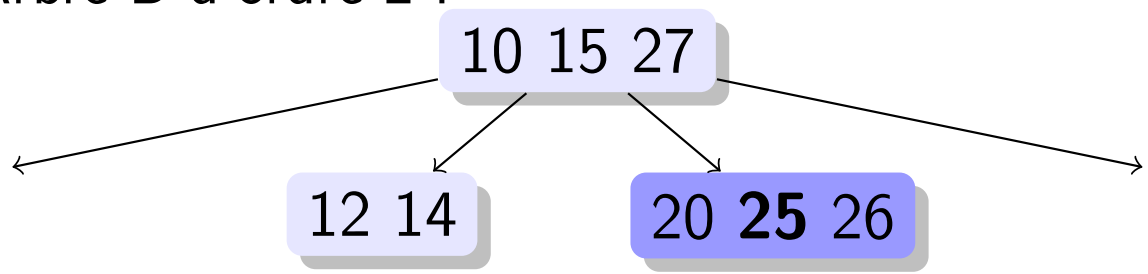
1. La suppression se fait toujours au niveau des feuilles
   ↪ Si la clé à supprimer n'est pas dans une feuille, alors la remplacer par la plus grande valeur des plus petites (ou plus petite valeur des plus grandes) et supprimer cette dernière
2. Si la suppression de la clé d'une feuille (récursivement d'un nœud) amène à avoir moins de m clés :
   1. Combinaison avec un nœud voisin (avant ou après)
   2. Descente de la clé associant ces deux nœuds (éclatement du nœud si nécessaire et donc remonté d'une nouvelle clé)

   ↪ la récursivité de ce principe peut ammener à diminuer la hauteur de l'arbre (par le haut)

INSA ROUEN NORMANDIE

Ex. cas 1 : très simple

Arbre-B d'ordre 2 :



Rappel : ici nombre de clés par nœud non racine > 1

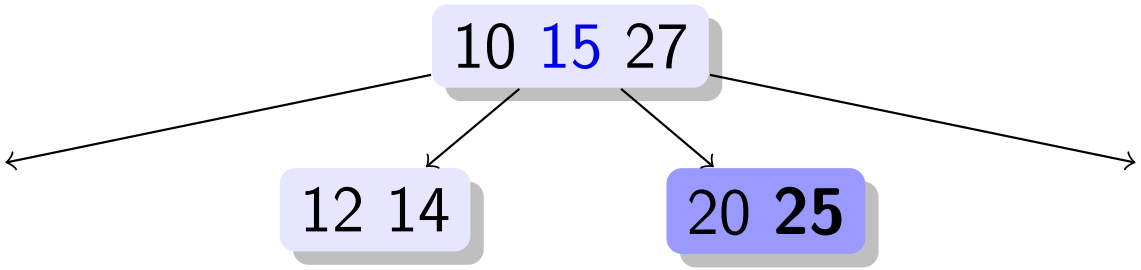
$\hookrightarrow$ Exemple : Suppression de la clé **25** ?

Méthode

1. Suppression de la valeur (décalage des valeurs dans le tableau)

Ex. cas 2 : simple 1 / 2

Arbre-B d'ordre 2 :



Rappel : ici nombre de clés par nœud non racine > 1

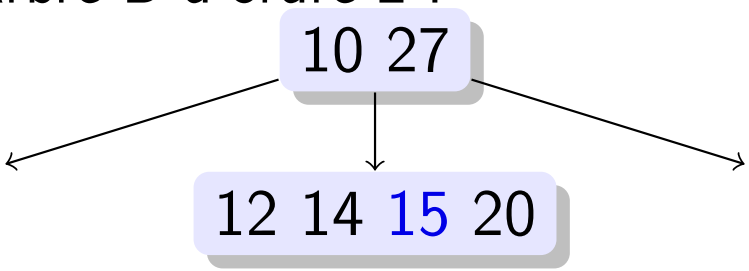
↪ EXEMPLE : Suppression de la clé **25** ?

Méthode

1. Combinaison avec un nœud voisin
2. Descente de la clé (ici 15)
3. Suppression du nœud

Ex. cas 2 : simple 2 / 2

Arbre-B d'ordre 2 :



Rappel : ici nombre de clés par nœud non racine > 1

↪ EXEMPLE : Suppression de la clé **25** ?

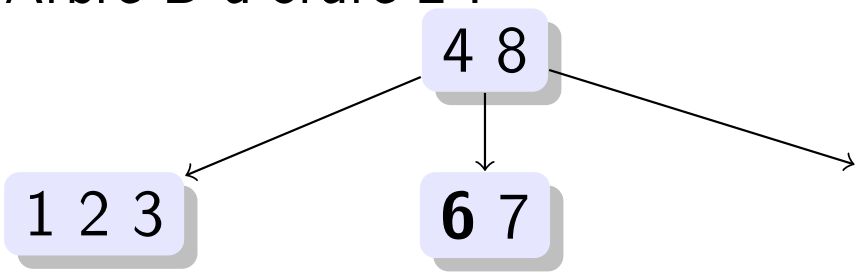
Méthode

1. Combinaison avec un nœud voisin ([12 14] et 20)
2. Descente de la clé médiane (ici 15)
3. Suppression du nœud

INSA ROUEN NORMANDIE

Ex. cas 3 : avec éclatement 1 / 2

Arbre-B d'ordre 2 :



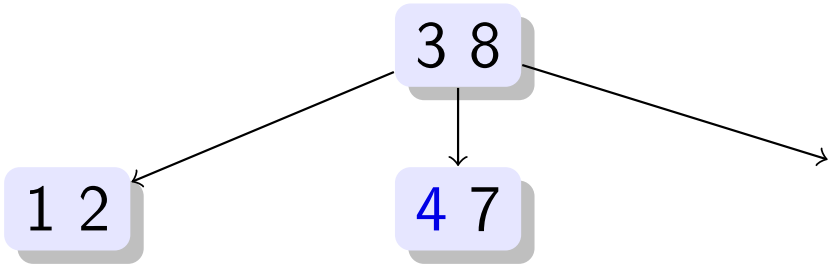
Rappel : ici nombre de clés par nœud non racine < 5 et > 1

$\hookrightarrow$ EXEMPLE : Suppression de la clé **6** ?

- Suppression clé **6** $\rightarrow$ Combinaison [1 2 3] et 7 + descente de la clé 4 au nœud fils
- Descente de la clé 4 au nœud fils $\rightarrow$ Redistribution + remontée de clé médiane (e.g. ici **3**)

Ex. cas 3 : avec éclatement 2 / 2

Arbre-B d'ordre 2 :



Rappel : ici nombre de clés par nœud non racine < 5

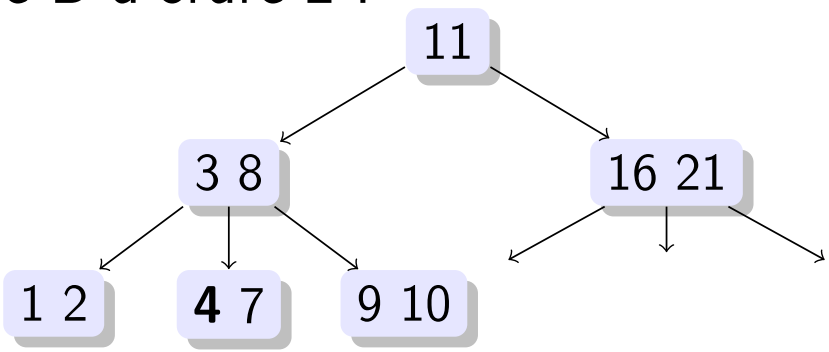
$\hookrightarrow$ EXEMPLE : Suppression de la clé **6** ?

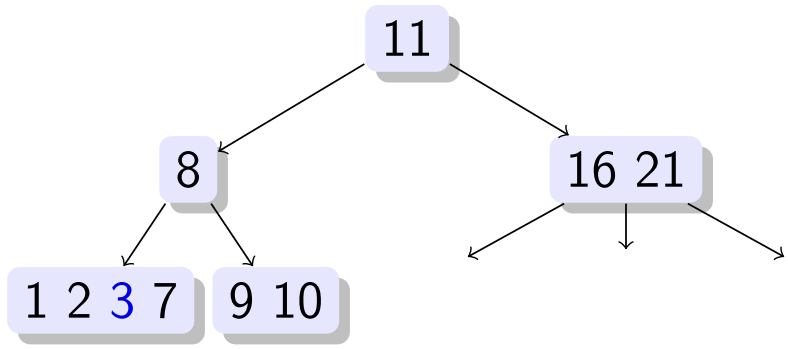
- Suppression clé **6** $\rightarrow$ Combinaison [1 2 3] et 7 + descente de la clé 4 au nœud fils
- Descente de la clé 4 au nœud fils $\rightarrow$ Redistribution + remontée de clé médiane (e.g. ici **3**)

INSA ROUEN NORMANDIE

Ex. cas 4 : avec un nombre de clés inférieurs à m 1 / 2

Arbre-B d'ordre 2 :



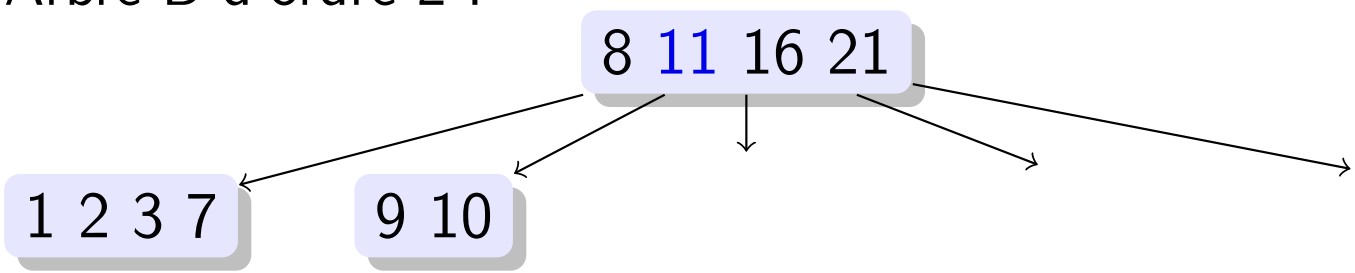


↪ EXEMPLE : Suppression de la clé **4** ?

- Combinaison ([1 2] et 7) + descente de la clé 3

Ex. cas 4 : avec un nombre de clés inférieurs à m 2 / 2

Arbre-B d'ordre 2 :



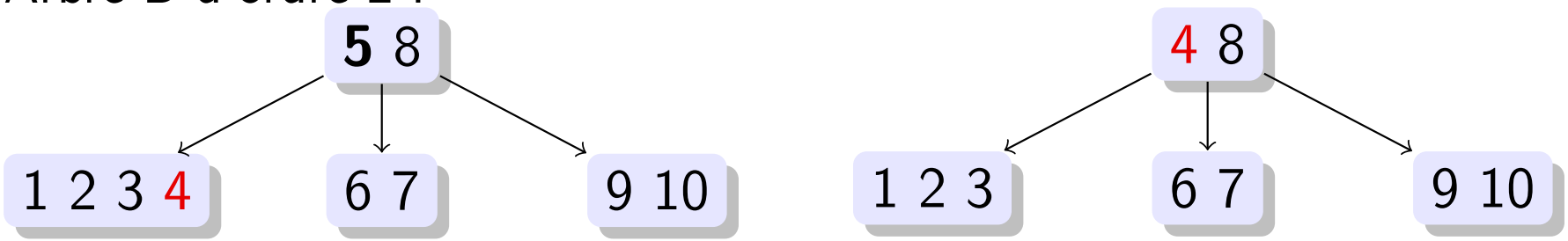
↪ EXEMPLE : Suppression de la clé **4** ?

- Combinaison + descente de la clé 3
- Combinaison (8 et [16 21]) + descente de la clé 11

↪ **Diminution d'une unité de la hauteur**

Ex. cas 5 : suppression non feuille

Arbre-B d'ordre 2 :



↪ EXEMPLE : Suppression de la clé **5** ?

Méthode

1. Recherche d'une clé adjacente A à la clé à supprimer → on choisit la plus grande du sous arbre gauche
2. Remplacement de la clé à supprimer par A
3. Suppression de la clé A du sous arbre gauche

INSA ROUEN NORMANDIE

Algorithme 1 / 4

Prérequis

On suppose posséder les fonctions/procédures suivantes :

- **fonction** plusGrandeValeur **(**a : ArbreB**) :** Valeur
  ⌊**précondition(s)** a ≠ **NIL**
- **fonction** positionCleDansNoeudRacine **(**a : ArbreB, c : Valeur**) : Entier**
  ⌊**précondition(s)** a ≠ **NIL**
  -1 si non présent
- **fonction** positionSsArbrePouvantContenirValeur **(**a : Arbre, c : Valeur**) : Naturel**
- **procédure** freres **(E** a : ArbreB, posSsArbre : **Naturel**,**S** frereG, frereD : ArbreB**)**
  ⌊**précondition(s)** a ≠ **NIL** et a^.sousArbres[posSsArbre] ≠ **NIL**
- **procédure** supprimerCleDansNoeudFeuille **(E/S** n : Noeud, **E** c : Valeur**)**
- **procédure** copierValeurs **(S** tDest : **Tableau[**1..MAX**] de** Valeur, **E** tSource : **Tableau[**1..MAX**] de** Valeur, indiceDebutDest, indiceDebutSource, nb : **NaturelNonNul)**
  ⌊**précondition(s)** indiceDebutSource+nb<MAX et indiceDebutDest+nb<MAX
- **procédure** decalerVersGaucheClesEtSsArbres **(E/S** n : Noeud, **E** aPartirDe : **NaturelNonNul**, nbCran : **NaturelNonNul)**

```
procédure supprimer (E/S a : ArbreB, E c : Valeur, ordre : NaturelNonNul)
debut
    a ← suppression(a,c,ordre,NIL,NIL,uneCle)
fin
```

Algorithme 2 / 4

Cas simples

```
fonction suppression (a : ArbreB, c : Valeur, ordre : NaturelNonNul, frereG, frereD : ArbreB, clePere : Valeur) : ArbreB
    Déclaration ...
debut
    si a = NIL alors
        retourner a
    sinon
        pos ⟵ positionCleDansNoeudRacine(a,c)
        si estUneFeuille(a) alors
            si pos = -1 alors
                retourner a
            sinon
                si frereG = NIL et frereD = NIL et a^.nbCles=1 alors
                    desallouer(a) // n'est possible que pour la racine
                sinon
                    Algo #1 : supprimer c dans une feuille
                finsi
            finsi
        sinon
            si pos = -1 alors
                posSsArbre ⟵ positionSsArbrePouvantContenirValeur(a,c)
            sinon
                cas # 5 des exemples
                cleRemplacement ⟵ plusGrandeValeur(a^.sousArbres[pos-1])
                a^.valeurs[pos] ⟵ cleRemplacement
                c ⟵ cleRemplacement
                posSsArbre ⟵ pos-1
            finsi
            Algo #2 : supprimer c dans un sous-arbre de rang posSsArbre
        finsi
    finsi
fin
```

Algorithme 3 / 4

Algo #1 : supprimer c dans une feuille

```
supprimerCleDansNoeudFeuille(a^,c)
si a^.nbCles≥ordre ou (frereG = NIL et frereD = NIL) alors
    retourner a // cas #1 des exemples
sinon
    // cas avec combinaison avec l'un des frere
    si frereG ≠ NIL alors
        copierValeurs(tab,frereG^.valeurs,1,1,frereG^.nbCles)
        tab[frereG^.nbCles+1] ← clePere
        copierValeurs(tab,a^.valeurs,frereG^.nbCles+2,1,a^.nbCles)
        nb ← 1+a^.nbCles+frereG^.nbCles
        desallouer(frereG)
    sinon
        copierValeurs(tab,a^.valeurs,1,1,a^.nbCles)
        tab[a^.nbCles+1] ← clePere
        copierValeurs(tab,frereD^.valeurs,a^.nbCles+2,1,frereD^.nbCles)
        nb ← 1+a^.nbCles+frereD^.nbCles
        desallouer(frereD)
    finsi
    res ← creerFeuille(tab,nb)
    desallouer(a)
    si nb>2*ordre alors
        retourner eclatement(res, ordre)
    finsi
    retourner res
finsi
```

Algorithme 4 / 4

Algo #2 : supprimer c dans le sous-arbre

```
freres(a,posSsArbre,frereG,frereD)
res ← suppression(a^.sousArbres[posSsArbre],c,ordre,frereG,frereD, a^.valeurs[
posSsArbre])
si res ≠ a^.sousArbres[posSsArbre] alors
    //cas ou il y a eu combinaison avec un frere
    si res^.nbCles=1 alors
        // cas #3 des exemples : il y a eu éclatement après combinaison
        a^.valeurs[posSsArbre] ← res^.valeurs[1]
        a^.sousArbres[posSsArbre-1] ← res^.sousArbres[0]
        a^.sousArbres[posSsArbre] ← res^.sousArbres[1]
    sinon
        // cas #2 des exemples
        decalerVersGaucheClesEtSsArbres(a^,posSsArbre,1)
        a^.sousArbres[posSsArbre] ← res
    finsi
finsi
retourner a
```

Exercice

Comment prendre en compte le cas #4 des exemples ?